NAME

iic - I2C generic I/O device driver

SYNOPSIS

device iic

#include <dev/iicbus/iic.h>

DESCRIPTION

The **iic** device driver provides generic I/O to any iicbus(4) instance. In order to control I2C devices, use */dev/iic?* with the following ioctls:

I2CSTART (*struct iiccmd*) Sends the start condition to the slave specified by the *slave* element to the bus. The *slave* element consists of a 7-bit address and a read/write bit (that is, a 7-bit address << 1 | r/w). A read operation is initiated when the read/write bit is set, or a write operation when it is cleared. All other elements are ignored. If successful, the file descriptor receives exclusive ownership of the underlying iicbus instance.

I2CRPTSTART (*struct iiccmd*) Sends the repeated start condition to the slave specified by the *slave* element to the bus. The slave address should be specified as in I2CSTART. All other elements are ignored. I2CSTART must have previously been issued on the same file descriptor.

I2CSTOP No argument is passed. Sends the stop condition to the bus. If I2CSTART was previously issued on the file descriptor, the current transaction is terminated and exclusive ownership of the underlying iicbus instance is released. Otherwise, no action is performed.

I2CRSTCARD (*struct iiccmd*) Resets the bus. The argument is completely ignored. This command does not require I2CSTART to have been previously issued on the file descriptor. If it was previously issued, exclusive ownership of the underlying iicbus instance is released.

I2CWRITE (*struct iiccmd*) Writes data to the iicbus(4). The bus must already be started by a previous I2CSTART on the file descriptor. The *slave* element is ignored. The *count* element is the number of bytes to write. The *last* element is a boolean flag. It must be zero when additional read commands will follow, or non-zero if this is the last command. The *buf* element is a pointer to the data to write to the bus.

I2CREAD (*struct iiccmd*) Reads data from the iicbus(4). The bus must already be started by a

previous I2CSTART on the file descriptor. The *slave* element is ignored. The *count* element is the number of bytes to read. The *last* element is a boolean flag. It must be zero when additional read commands will follow, or non-zero if this is the last command. The *buf* element is a pointer to where to store the data read from the bus. Short reads on the bus produce undefined results.

I2CRDWR (*struct iic_rdwr_data*) Generic read/write interface. Allows for an arbitrary number of commands to be sent to an arbitrary number of devices on the bus. Any previous transaction started by I2CSTART must be terminated by I2CSTOP or I2CRSTCARD before I2CRDWR can be issued on the same file descriptor. A read transfer is specified if IIC_M_RD is set in *flags*. Otherwise the transfer is a write transfer. The *slave* element specifies the 7-bit address with the read/write bit for the transfer. The read/write bit will be handled by the iicbus stack based on the specified transfer operation. The *len* element is the number of (*struct iic_msg*) messages encoded on (*struct iic_rdwr_data*). The *buf* element is a buffer for that data. This ioctl is intended to be Linux compatible.

I2CSADDR (*uint8_t*) Associate the specified address with the file descriptor for use by subsequent read(2) or write(2) calls. The argument is an 8-bit address (that is, a 7-bit address << 1). The read/write bit in the least-significant position is ignored. Any subsequent read or write operation will set or clear that bit as needed.

The following data structures are defined in <*dev/iicbus/iic.h*> and referenced above:

```
struct iiccmd {
        u_char slave;
        int count;
        int last;
        char *buf;
};

/* Designed to be compatible with linux's struct i2c_msg */
struct iic_msg
{
        uint16_t   slave;
        uint16_t   flags;
#define   IIC_M_WR        0       /* Fake flag for write */
#define   IIC_M_RD        0x0001  /* read vs write */
#define   IIC_M_NOSTOP    0x0002  /* do not send a I2C stop after message */
#define   IIC_M_NOSTART   0x0004  /* do not send a I2C start before message */
```

```
        uint16_t  len;      /* msg length */
        uint8_t *  buf;
};

struct iic_rdwr_data {
        struct iic_msg *msgs;
        uint32_t nmsgs;
};
```

It is also possible to use read(2) or write(2), in which case the I2C start/stop handshake is managed by iicbus(4). The address used for the read/write operation is the one passed to the most recent I2CSTART ioctl(2) or I2CSADDR ioctl(2) on the open */dev/iic?* file descriptor. Closing the file descriptor clears any addressing state established by a previous I2CSTART or I2CSADDR, stops any transaction established by a not-yet-terminated I2CSTART, and releases iicbus ownership. Because addressing state is stored on a per-file-descriptor basis, it is permissible for multiple file descriptors to be simultaneously open on the same */dev/iic?* device. Concurrent transactions on those descriptors are synchronized by the exclusive-ownership requests issued to the underlying iicbus instance.

SEE ALSO

ioctl(2), read(2), write(2), iicbus(4)

HISTORY

The **iic** manual page first appeared in FreeBSD 3.0.

AUTHORS

This manual page was written by Nicolas Souchu and M. Warner Losh.